INFORMATION AND COMPUTATION 117, 136–150 (1995)

Suprema of Open and Closed Formulas and Their Application to Resolution*

MARCO BELLIA† AND M. EUGENIA OCCHIUTO

Dipartimento di Informatica, Università di Pisa, Corso Italia 40, 56100 Pisa, Italy
E-mail: occhiuto@di.unipi.it

We discuss an algebra of open formulas, SOAF. The algebra is equipped with the operator *mgi-o*, which computes suprema of sets of open formulas, ordered according to a weak form of instance ordering. Rules to compute such points are provided. These rules supply a computation procedure to unify open formulas. Next, we consider an algebra of closed formulas, SCAF. It has an operator *mgi-c*, which computes suprema of set of formulas ordered according to the instance preordering. We give rules to compute such points. These rules supply a computation procedure to unify closed formulas. Relations between these operators and unification in open and closed formulas, and weak unification in closed formulas, are addressed. Also, we consider clausal formulas and apply these considerations to the resolution process. This provides a rule alternative to resolution and shows that resolvents can be expressed and effectively obtained as suprema of closed formulas. The new rule need not resort to renaming, unification, and instantiation as first class operations. Finally, applications to resolution in Horn clause logic and logic programming are discussed. This leads to a more compact theory of logic programming, in which, for instance, the domain of substitutions is eliminated.

1. INTRODUCTION

In 1965 Robinson introduced unification as the kernel of first order resolution (Robinson, 1965). Unification provides resolution with a substitution rule (Huet, 1972) that, in the case of non-satisfiable theories, reduces the search space and makes resolution finite. However, unification has several drawbacks. First, it is defined on sets of open formulas: Occurrences of the same variable symbol in different formulas stand for the same variable. For instance, when unification is applied to the set of formulas $I = \{(A(fX)), (AX)\}$, the variable symbol X occurring in the formula $(A(fX))$ is identified with the X occurring in the formula (AX). Unfortunately, clauses are closed formulas. Robinson gives a solution of the problem considering closed formulas as the set of variants of open formulas. In this way, unification in closed formulas is solved through unification on suitable, but arbitrary, variants of them. It is necessary at this point to introduce a *renaming* operation. Variables are renamed so that variable symbols occurring in variants of distinct closed formulas are distinct. The cost of this operation is not in itself relevant, but yields complex techniques of standardizing (Apt, 1990; Ko and Nadel, 1991) in the theory of logic programming. (Asperti and Martini, 1989; Bellia *et al.*, 1990) avoid standardizing by using algebras of variable free formulas.

Another problem arising from unification is the need for substitution functions. Unification computes the most general substitution (function). This yields resolution to deal with the structure of substitutions besides that of formulas. Unfortunately, the algebra of substitutions has a poor structure; for instance, its notion of equivalence does not correspond to any notion of congruence (Palamidessi, 1990). Moreover, it does not always allow characterization of the most general substitution (Eder, 1985; Lassez *et al.*, 1988). For these reasons, it is necessary to consider a strict subclass of substitutions, the class of idempotent substitutions. This class, unfortunately, is not a subalgebra; in fact, it is not closed under composition.

Finally, though we restrict to idempotent substitutions, using idempotent mgu to unify closed formulas introduces the following problem. Let I be defined as above. Then $I^Y = \{(A(fY)), (AX)\}$ is a variant of I and $\mathrm{mgu}(I^Y) = \{X \leftarrow (fY)\}$. On the other hand, $I^Z = \{(A(fZ)), (AX)\}$ is also a variant of I and $\mathrm{mgu}(I^Z) = \{X \leftarrow (fZ)\}$. But $\mathrm{mgu}(I^Y)$ and $\mathrm{mgu}(I^Z)$ are incomparable in the sense that neither of them is smaller than the other in the idempotent subsitution lattice $I/\sim$ (Eder, 1985). A solution can be obtained using weak unification on closed formulas (Eder, 1985). Unfortunately, weak unifiers too do not always have a most general weak unifier. In addition, upper bounds of weak unifiers are not necessarily weak unifiers (Bellia and Occhiuto, 1991).

* Partially supported by ESPRIT Basic Research Action P3020 ("Integration"), and by CNR, Progetto Finalizzato "Sistemi Informatici e Calcolo Parallelo."

† On leave from Dipartimento di Matematica e Applicazioni, Università di Napoli.

0890-5401/95 $6.00

We will return to weak unification in Proposition 30. A more general solution seems to be provided by restricted unification (Baader, 1990), which is not very clearly motivated for resolution. We will also return to restricted unification in Proposition 24.

Also, from the implementation point of view, substitutions are not satisfactory. For instance, almost all implementations of logic languages, including the ingenious WAM (Warren, 1983), do not provide any specific structure for substitutions, because this is considered uselessly expensive. The correspondence between the theory and its implementation is highly compromised: Implementations are sometimes obscure, correct implementations are hard to derive, and proving correctness is difficult and not completely satisfactory.

These considerations may suggest that unification is too subtle an operation and, anyway, not well suited to resolution. This leads us to define a new rule alternative to resolution that does not require substitutions. The rule satisfies all the following points: (i) it suits properly from the theoretical point of view; (ii) it mantains an algorithmic nature; (iii) it is acceptable from the efficiency point of view. Clauses are presented as closed formulas of a structure SCAF. Then, this rule computes resolvents as suprema of formulas in SCAF.

Reynolds in (Reynolds, 1970) introduced an algebra of closed atomic formulas, CAF. They form a lattice, with instance as partial ordering, greatest common instance, gci, as meet operation, and least common generalization, lcg, as joint operation. To compute gci Reynolds uses unification on suitable variants of the formulas. We revisit the structure of atomic formulas, restricting the instance ordering and extending the formulas to sequences of atomic formulas. Thus, we obtain an algebra of sequences of open atomic formulas, SOAF. SOAF is still a lattice under this partial ordering. Moreover, the equivalence classes of SOAF under the variant relation form a lattice, SCAF, in which CAF is embedded. Rules to infer suprema of SOAF, mgi-o, and SCAF, mgi-c, are included. The gci of CAF can be computed using mgi-c as a first class operation avoiding renaming, unification, and instantiation. We then discuss the relationships between mgi-o and mgi-c, on one hand, and unification, weak unification, and restricted unification, on the other hand.

2. SUBSTITUTIONS

In this section we summarize the main notions and properties related to unification that will be used in the sequel. For more complete insight see (Eder, 1985; Lassez *et al.*, 1988; Knight, 1989; Kirchner, 1990).

Given a set M of Σ-terms, a unification problem is finding the minimum substitution θ such that all elements in M unify; that is, $M\theta$ is a singleton. θ is called the most general unifier, mgu for short. There are other presentations of unification problems (Lassez *et al.*, 1988), but all are equivalent to the previous one. Even the extension given by Eder (Eder, 1985) in which M is a multi-set is not substantially different from this one. Therefore, unification problems deal with two kinds of structures: the structure of Σ-terms and the structure of substitutions.

DEFINITION 1. *Σ-terms* are the terms of a free Σ-algebra $(\Sigma, \mathcal{V})$, where $\Sigma = \bigcup \Sigma_i$ is a set of Σ-operators (functors) indexed by the arity i, i.e., $f \in \Sigma_{\text{arity}(f)}$, and $\mathcal{V}$ is a (possibly denumerable) set of variables. The set of Σ-terms, $\Sigma(\mathcal{V})$, is the closure of $\mathcal{V}$ under finitely many applications of operators to Σ-terms: $(f\, t_1 \cdots t_n)$, or simply f if $n = 0$, denotes the application of the n-ary operator f to terms $t_1 \cdots t_n$. Σ-terms are often considered as finite ordered trees, labelled with operators from Σ and variables from $\mathcal{V}$. Usually Σ-terms are equipped with the following operators, for Σ-terms h, g and path p: Subterm *selection*, h/p, subterm *replacement*, $h\{p \leftarrow g\}$, and the operator to compute the *set of variables* occurring in a Σ-term h, $\text{Var}(h)$.

DEFINITION 2. A *substitution* θ is a finite function that maps variables into Σ-terms, i.e., $\theta \in \mathcal{V} \to \Sigma(\mathcal{V})$. The domain, $D(\theta)$, is the finite set $\{x \mid x\theta \neq x\}$. The range, $R(\theta)$, is the finite set of variables occurring in the terms $x\theta$, i.e., $\{\text{Var}(x\theta) \mid x \in D(\theta)\}$. In the sequel, we call the set of all such substitutions *Subst*.

Let t be a term and let θ be a substitution. Then the *instantiation* $t\theta$ is defined in the following way: $t\theta = \theta(t)$ if $t \in \mathcal{V}$ and $(f\, t_1 \cdots t_n)\,\theta = (f\, t_1\theta \cdots t_n\theta)$ (or simply f if $n = 0$) if $t \notin \mathcal{V}$. If M is a set of Σ-terms then $M\theta$ denotes the set $\{t\theta \mid t \in M\}$. Instantiation makes it possible to define a preordering relation $\leqslant$ on $\Sigma(\mathcal{V})$, such that $h \leqslant h\theta$. Then we have an equivalence relation in the obvious way, $h \sim g$ iff $h \leqslant g$ and $g \leqslant h$. $[\Sigma(\mathcal{V})/\sim, \leqslant]$, completed with a maximum $\top$, is a complete lattice having the variable term as bottom ((Reynolds, 1970; Huet, 1976), where $\leqslant$ is reverted and then completed with a minimum $\bot$). In Section 5 on closed formulas, we revisit such a structure.

Substitution composition provides an ordering relation $\leqslant_\circ$ on substitutions, namely $\sigma \leqslant_\circ \sigma\rho$. But we have the following negative result.

PROPOSITION 3. Subst *completed with* $\top$ *as top element and having identity* ε *as bottom element is not a lattice* (*Eder*, 1984).

It is necessary to restrict to the class $I/\sim$ of the idempotent substitutions modulo the equivalence relation $\sim$ to have a lattice that is embedded in Subst: Two subsitutions σ *and* τ *are* $\sim$*-equivalent* iff $\sigma \leqslant_\circ \tau$ and $\tau \leqslant_\circ \sigma$. As mentioned above, the class of idempotent substitutions is not closed under the composition: Composition of two

idempotent substitutions is not, in general, an idempotent substitution. Substitution composition is an associative but not commutative operator. There is only one very special case in which the composition of substitutions is, in fact, commutative. This is explained by the following lemma.

LEMMA 4. *Let θ and ρ be substitutions; then $\theta\rho = \rho\theta$ iff*

- $D(\rho) \cap R(\theta) = \{\ \}$ *and* $D(\theta) \cap R(\rho) = \{\ \}$
- $\forall x \in D(\theta)$ *either* $x \notin D(\rho)$ *or* $x\theta = x\rho$.

Proof. The second condition guarantees that there are not two different bindings for the same variable in θ and ρ. Otherwise, the composition would contain the binding of the substitution that was applied first. The first condition guarantees that the bindings in the substitution first applied are not modified by the composition with the second substitution. ∎

DEFINITION 5. A *permutation* is a substitution $\theta \in \mathcal{V} \to \mathcal{V}$ s.t. $D(\theta) = R(\theta)$.

LEMMA 6. *Let σ and σ' be two substitutions such that $\sigma \sim \sigma'$. Then there are two permutations τ and τ' such that $\sigma' = \sigma\tau$ and $\sigma = \sigma'\tau'$ (Eder, 1985).*

DEFINITION 7. *Let $W \subseteq \mathcal{V}$.* A *W-renaming* is a substitution $\theta \in \mathcal{V} \to \mathcal{V}$ such that $D(\theta) = W$ and θ is injective on W.

DEFINITION 8. We define a *renaming* to be an idempotent W-renaming.

Note that not all substitutions have an inverse, not even all $\mathcal{V} \to \mathcal{V}$ substitutions. We introduce a weak inverse ρ^- which always exists for $\mathcal{V} \to \mathcal{V}$ substitutions.

DEFINITION 9. Let $\rho \in \mathcal{V} \to \mathcal{V}$. We define ρ^- to be a substitution s.t. $(x\rho^-)\,\rho = x$ for each $x \in R(\rho)$, and $x\rho^- = x$ for each $x \in \mathcal{V} - R(\rho)$.

When ρ is a renaming, ρ^- is unique.

LEMMA 10. *Let ρ be a renaming. Then* (a) $(\rho^-)^- = \rho$ *and* (b) $\rho\rho^- = \rho^-$.

Proof. (a) by definition of renaming, ρ is injective from $D(\rho)$ onto $R(\rho)$. (b) $\forall x \in \mathcal{V} - D(\rho)$, $x\rho\rho^- = x\rho^-$, and $\forall x \in D(\rho)$, $x\rho\rho^- = x$ by definition of ρ^-. But for ρ idempotent, if $x \in D(\rho)$ then $x \notin R(\rho)$, hence $x\rho^- = x$. ∎

LEMMA 11. *Let θ be a substitution and let ρ be a renaming with $D(\rho) \subseteq R(\theta)$ and $R(\rho) \subseteq D(\theta) - R(\theta)$. Then $\theta\rho \sim \theta$.*

Proof. $\theta \leqslant_o \theta\rho$ by definition of $\leqslant_o$. Moreover, $\theta\rho\rho^- = \theta$. In fact, $\theta\rho\rho^- = \theta\rho^-$ by Lemma 10. Moreover, $\forall x \in D(\theta)$, $x\theta\rho^- = x\theta$ since $D(\rho^-) \subseteq D(\theta) - R(\theta)$, and $\forall x \in \mathcal{V} - D(\theta)$, $x\theta\rho^- = x\rho^- = x$. ∎

LEMMA 12. *For each substitution σ, let σ^{-1} be a function from variables into sets of variables s.t. $\sigma^{-1}(x) = \{y \in \mathcal{V} \mid y\sigma = x\}$. Let I^σ be a set of $\mathcal{V} \to \mathcal{V}$ functions s.t. $I^\sigma = \{\rho \in \mathcal{V} \to \mathcal{V} \mid x\rho \in \sigma^{-1}(x)$ for each $x \in D(\sigma) \cap R(\sigma)$ and $x\rho = x$ otherwise$\}$. Then*

(a) *for each $\rho \in I^\sigma$, ρ is a substitution and $\sigma\rho$ is an idempotent substitution such that $\sigma\rho \sim \sigma$;*

(b) *every idempotent substitution θ that is equivalent to σ can be obtained as $\sigma\rho$ for $\rho \in I^\sigma$;*

(c) *every substitution $\theta = \sigma\rho$ for $\rho \in I^\sigma$ is s.t. $\theta\rho^- = \sigma$.*

Proof. (a) and (b) follow straightforwardly from Lemma 3.4 in (Eder, 1985). For (c) we note that ρ^- is by construction the restriction of $\sigma R(\rho)$; i.e., $\rho^- = \sigma \upharpoonright R(\rho)$. Moreover, for each $x \in D(\sigma)$, if $x \notin D(\theta)$ then $x \in R(\rho)$ and $x\theta\rho^- = x\rho^- = x\sigma \upharpoonright R(\rho) = x\sigma$. If $x \in D(\theta)$ then $x\theta\rho^- = x\sigma\rho\rho^-$. Let $y \in \mathrm{Var}(x\sigma)$. Hence $y \in R(\sigma)$. If $y \notin D(\sigma) \cap R(\sigma)$ then $y \notin D(\sigma)$, $y\rho = y$ and $y\rho^- = y$. If $y \in D(\sigma) \cap R(\sigma)$ then $y\rho = z \in \sigma^{-1}(y)$, $z\rho^- = z\sigma = y$. Hence, $y\rho\rho^- = y$ and $x\sigma\rho\rho^- = x$. ∎

A useful operation on substitutions is restriction. Let $V \subset \mathcal{V}$; we then define $x\theta \upharpoonright V = x\theta$, for each $x \in V \cap D(\theta)$, and $x\theta \upharpoonright V = x$, for each $x \notin V \cap D(\theta)$.

LEMMA 13. *Let $\theta \sim \sigma$ be idempotent substitutions, and let τ, ρ be such that $\theta = \sigma\rho$ and $\sigma = \theta\tau$. Then $\tau \upharpoonright R(\theta)$ and $\rho \upharpoonright R(\sigma)$ are renamings. Moreover, $\tau \upharpoonright R(\theta)^- = \rho \upharpoonright R(\sigma)$.*

Proof. Since $\theta \sim \sigma$, τ and ρ are injective. Moreover, $D(\tau \upharpoonright R(\theta)) \subseteq D(\sigma)$, $R(\tau \upharpoonright R(\theta)) \subseteq R(\sigma)$, $D(\sigma) \cap R(\sigma) = \{\ \}$, and hence $\tau \upharpoonright R(\theta)$ is idempotent and so is $\rho \upharpoonright R(\sigma)$. Hence, $\tau \upharpoonright R(\theta)$ and $\rho \upharpoonright R(\sigma)$ are renamings. Finally, for each $x \in R(\theta)$, $x\tau\rho = x\tau \upharpoonright R(\theta)\,\rho \upharpoonright R(\sigma) = x$. Hence $\tau \upharpoonright R(\theta) = \rho \upharpoonright R(\sigma)^-$ and also $\tau \upharpoonright R(\theta)^- = \rho \upharpoonright R(\sigma)$, by Lemma 10(c). ∎

Actually, most general unifiers are characterized in the Eder lattice $[I/\sim, \leqslant_o]$ of the idempotent substitutions modulo $\sim$, completed with the maximum substitution $\top$. Given a set M of Σ-terms, mgu(M) computes the idempotent substitution that is the greatest lower bound, glb, of all the idempotent substitutions that are unifiers of M, i.e., $\mathrm{mgu}(M) = \mathrm{glb}\{\theta \in I/\sim \mid M\theta$ is singleton$\}$. When M is not unifiable, $\mathrm{mgu}(M) = \top$.

PROPOSITION 14 (Lassez et al., 1988, Proposition 16, p. 606). *For each most general unifier there is an equivalent idempotent most general unifier.*

PROPOSITION 15 (Lassez et al., 1988, Corollary 10, p. 606). *If μ and σ are two different most general unifiers of the same set M, and are such that μ is non-idempotent and σ is idempotent, then $|D(\mu)| > |D(\sigma)|$, where $|X|$ is the cardinality of the set X.*

3. AN ALGEBRA FOR OPEN FORMULAS: SOAF

DEFINITION 16. Let P be a (finite) set of predication symbols (indexed by their arity), let Σ be a (non-trivial finite) set of function symbols (indexed by their arity), and let $\mathcal{V}$ be a denumerable set of variable symbols. Then $\Sigma(P, \mathcal{V})$ is the minimum set containing:

(i) Ω and Top

(ii) all the finite sequences $u_1 \cdots u_n$ $(n \geqslant 1)$ with $u_i \in P(\Sigma(\mathcal{V})) \cup \Sigma(\mathcal{V})$, where $P(\Sigma(\mathcal{V}))$ is the set of atomic formulas with symbols in P and terms in $\Sigma(\mathcal{V})$.

$\Sigma(P, \mathcal{V})$ includes Σ-terms and atomic formulas as one-component sequences, and contains all the finite sequences of terms and atomic formulas. Components $u_1, \ldots, u_n$ of the sequence $u_1 \cdots u_n$ are called *atoms*. We extend to $\Sigma(P, \mathcal{V})$ the already defined operators on $\Sigma(\mathcal{V})$. In partiuclar, $U/0$ selects the first atom of the sequence U and so on, $U/i-1$ selects the ith atom. In addition, we introduce the following manipulation operators. Let U be a sequence:

Tup(U) computes the length of U;

Paths(U, x) $= \{i.p \mid p \in \text{Paths}(U/i, x)\}$ computes the set of paths in the sequence U at the variable x;

Ph(U, x) returns an arbitrary path in Paths(U, x) when Paths (U, x) is not empty;

Main(u) returns the main (function or predication) symbol of the atom u.

Moreover, for $i \in [0, \text{tup}(U)-1]$:

Remove(U, i) removes the main symbol of the ith atom of U, i.e., replaces in U, $U/i = (f\, h_1 \cdots h_n)$ with the sequence $h_1 \cdots h_n$;

Insert(U, i, f) inserts in U the symbol f as the main symbol of $U/i \cdots U/\text{arity}(f)$, i.e., replaces in U the sequence $U/i \cdots U/\text{arity}(f)$ with the atom $(f\, U/i \cdots U/\text{arity}(f))$—hence $U = \text{Insert}(\text{Remove}(U, i), i, \text{Main}(U/i))$.

All the previous operators are extended to sets, when needed. The examples contained in this paper adopt the usual convention of denoting function symbols with lower letters, and variable and predication symbols with capital letters.

EXAMPLE. Let A, B be 2-arity predications, f a 2-arity function, g a 1-arity function, a, b 0-arity functions, and X, Y, Z variables. Then $U = (fXY)(ga)(B(gX)Z)$, and $V = (g(fab))X$ are sequences in $\Sigma(P, \mathcal{V})$. Moreover,

$$\text{Tup}(U) = 3, \quad \text{Paths}(U, X) = \{0.1, 2.1.1\},$$

$$\text{Main}(U/1) = g, \quad \text{Remove}(U, 2) = (fXY)(ga)(gX)Z,$$

$$\text{Insert}(U, 0, A) = (A(fXY)(ga))(B(gX)Z),$$

$$U\{0 \leftarrow X\} = X(ga)(B(gX)Z),$$

$$\text{Paths}(\{U, V\}, \{X, Y\}) = \{0.1, 1, 2.1.1, 0.2\}.$$

In this section, sequences in $\Sigma(P, \mathcal{V})$ are considered as first order open formulas, syntactically sequences of atomic formulas. Open formulas are characterized by the presence of quantifier free variables. For our purposes, all variables occurring in an open formula are quantifier free variables. Thus open formulas can be considered as syntactic components of closed formulas. A context allows us to compare distinct open formulas. We give a partial ordering on open formulas, using substitutions as contexts.

DEFINITION 17. Let $\leqslant 1$ be a total ordering on $\mathcal{V}$. We define Sub $\subset$ Subst such that $\theta \in$ Sub iff $\forall x$, if $x\theta \in \mathcal{V}$ then $x \leqslant 1\ x\theta$.

LEMMA 18. (a) Sub *is closed* w.r.t. *composition and restriction*; i.e., $\forall \theta, \rho \in$ Sub, $\theta\rho \in$ Sub, *and* $\forall V \subseteq \mathcal{V}$, $\theta \uparrow V \in$ Sub;

(b) *Renamings are only partially included in* Sub; *in particular, for each renaming θ in* Sub, θ^- *is not in* Sub. *Permutations are not included at all (except for ε).*

(c) *Composition preordering $\leqslant_o$ is a partial ordering on* Sub; *i.e., for each $\theta, \rho \in$ Sub, if $\theta \leqslant_o \rho$ and $\rho \leqslant_o \theta$ then $\theta = \rho$.*

Proof. (a) $\forall x \in D(\theta\rho)$ s.t. $x\theta\rho \in \mathcal{V}$, $x \leqslant 1\ x\theta$ and $x\theta \leqslant 1\ (x\theta)\,\rho$.

(b) Let $\theta \in$ Sub be a renaming. Then $x\theta^- \leqslant 1\ x$, and $\theta^- \notin$ Sub. Moreover, let θ be a permutation. We have that $D(\theta) = R(\theta)$ is finite, hence $\text{lub}(D(\theta)) \in D(\theta)$, and if $x = \text{lub}(D(\theta))$, then $x\theta \leqslant 1\ x$.

(c) Let $\theta\sigma = \rho$ and $\theta = \rho\tau$. Then by Lemma 13, $\sigma \uparrow R(\theta)$ and $\tau \uparrow R(\rho)$ are renamings and $\sigma \uparrow R(\theta)^- = \tau \uparrow R(\rho)$. But because of (b) above, $\tau \uparrow R(\rho) \in$ Sub iff it is ε. ∎

It is worth nothing that (c) in Lemma 18 means that if $\theta, \sigma \in$ Sub $\subset$ Subst and are s.t. $\theta \leqslant_o \sigma$ in Subst, then it does not follow that $\theta \leqslant_o \sigma$ in Sub. The converse obviously holds. Hence, Sub is embedded into Subst; i.e., (a) Sub is included in Subst and (b) ordering is preserved. This yields the following embedding relationships among $I/\sim$, Sub, and Subst.

PROPOSITION 19. *$I/\sim$ is embedded into* Sub *which is embedded into* Subst.

Proof. We need only prove that $I/\sim$ is embedded into Sub. We separately prove that (a) $I/\sim$ is included in Sub and (b) the ordering in $I/\sim$ is preserved in Sub. (a) Let $\theta \in I$ and $\theta \notin$ Sub. Then $\exists V \subseteq D(\theta)$ such that $V\theta \subseteq R(\theta) \subset \mathcal{V}$ and $\theta \uparrow (D(\theta) - V) \in$ Sub. Let ρ such that $x\rho = x$ for each $x \in \mathcal{V} - V\theta$, and $x\rho = \text{lub}\{y \mid y\theta = x\} \in D(\theta)$ for each $x \in V\theta$. Then ρ is a renaming on $V\theta \subseteq R(\theta)$. Moreover, $\theta\rho \in$ Sub and $D(\rho) \subseteq R(\theta)$ and $R(\rho) \subseteq D(\theta) - R(\theta)$. Hence, because of Lemma 11, $\theta\rho \sim \theta$. (b) Let ρ be s.t. $\sigma = \theta\rho$. Then $\rho \uparrow R(\theta)$ is in Sub and is such that $\sigma = \theta(\rho \uparrow R(\theta))$ ∎

Note that this guarantees that Sub contains enough functions to express unifiers and most general unifiers. However, Sub is not a lattice. This is shown by the following example that Eder uses to prove that also Subst is not a lattice.

EDER'S EXAMPLE (Eder, 1985, Example 2.7, p. 34). Let $\sigma, \theta \in$ Sub s.t.

$$\sigma = \{X \leftarrow (fX(fYZ)),\ Y \leftarrow (fX(fYZ)),\ Z \leftarrow (fX(fYZ))\} \quad \text{and}$$

$$\theta = \{X \leftarrow (f(fXY)\,Z),\ Y \leftarrow (f(fXY)\,Z),\ Z \leftarrow (f(fXY)\,Z)\}.$$

Then $\{\sigma, \theta\}$ is bounded from above but has no supremum.

THEOREM 20. *Let* MGU(Γ) *for $\Gamma \in I/\sim$ be the set of all the substitutions in* Subst *that are equivalent to Γ. Then* MGU(Γ) $\cap$ Sub *contains exactly one idempotent θ. Moreover, θ is smaller, in* Sub, *than any other substitution σ in* MGU(Γ) $\cap$ Sub.

Proof. Let θ, σ be both idempotent and such that $\theta \sim \sigma$ in Subst. Then by Lemma 13, τ and ρ exist s.t. $\tau \uparrow R(\theta) = \rho \uparrow R(\sigma)$. Because θ, σ are idempotent the following hold: $\theta \uparrow R(\sigma) = \rho \uparrow R(\sigma)$ and $\sigma \uparrow R(\theta) = \tau \uparrow R(\theta)$. But by (b) of Lemma 18, $\tau \uparrow R(\theta) \notin$ Sub when θ (hence, $\rho \uparrow R(\sigma)$) is in Sub and by (a) of the same lemma, $\sigma \notin$ Sub.

Finally, for each non-idempotent σ such that $\theta \sim \sigma$ in Subst, if θ, σ are both in Sub then $\theta \leqslant_0 \sigma$. But $\sigma \leqslant_0 \theta$ is not true in Sub. In fact, let $\theta = \sigma\rho$ for $\rho \in I^\sigma$ as in Lemma 12. Then $\sigma \uparrow D(\rho^-) = \rho^-$ hence $\rho^- \in$ Sub. By (c) of Lemma 12, $\sigma = \theta\rho^-$ and $\theta \leqslant_0 \sigma$ holds in Sub. Hence, $\rho \notin$ Sub due to (b) of Lemma 18. For any other ρ' s.t. $\theta = \sigma\rho'$ we note that ρ' is an extension of ρ, hence $\rho' \notin$ Sub. Hence $\sigma \leqslant_0 \theta$ does not hold in Sub. ∎

An important consequence is that although Sub is not a lattice, if we consider only unifiers, they are bounded from below and have a greatest lower bound in Sub. Hence, mgu is univocally charachterized in Sub as the greatest lower bound of all unifiers, idempotent and not idempotent, which are in Sub. As in Eder's characterization, mgu is an idempotent substitution. Consider, for instance, the set $I^V = \{(A(fY)), (AX)\}$ in the introductory example. $\{X \leftarrow (fY)\}$ is the unique idempotent mgu(I^V) belonging to Sub. Moreover, any other unifier of I^V that is a most general unifier in Subst either is not in Sub or is greater than $\{X \leftarrow (fY)\}$. For instance, $\{X \leftarrow (fX),\ Y \leftarrow X\}$ is a unifier of I^V and, in Subst, also a most general unifier. If it belongs to Sub, then $\{X \leftarrow (fX),\ Y \leftarrow X\}$ can be obtained from $\{X \leftarrow (fY)\}\{Y \leftarrow X\}$; assuming $Y \leqslant 1\ X$, then $\{Y \leftarrow X\}$ is also in Sub. Hence $\{X \leftarrow (fY)\} \leqslant_0 \{X \leftarrow (fX),\ Y \leftarrow X\}$ holds in Sub. But the converse does not. In fact, $\{X \leftarrow (fY)\}$ cannot be obtained from $\{X \leftarrow (fX),\ Y \leftarrow X\}\{X \leftarrow Y\}$ since $Y \leqslant 1\ X$; thus $\{X \leftarrow Y\}$ cannot belong to Sub.

3.1. *A Lattice for Open Formulas*

DEFINITION 21. We extend $\leqslant 1$ on the product $\Sigma(P, \mathcal{V}) \times$ Sub in the following way:

(a) $\langle \Omega, \varepsilon \rangle \leqslant 1 \langle U, \theta \rangle$, for each $U \in \Sigma(P, \mathcal{V})$ and $\theta \in$ Sub;

(b) $\langle U, \theta \rangle \leqslant 1 \langle \text{Top}, \top \rangle$, for each $U \in \Sigma(P, \mathcal{V})$ and $\theta \in$ Sub;

(c) $\langle U, \theta \rangle \leqslant 1 \langle U\theta', \theta\theta' \rangle$, for each $U \in \Sigma(P, \mathcal{V})$ and $\theta, \theta' \in$ Sub.

PROPOSITION 22. *$[\Sigma(P, \mathcal{V}) \times \text{Sub},\ \leqslant 1]$ is a partially ordered set. Moreover $\leqslant 1$ is homeomorphic, i.e., for each U and Z with same tupling and $q \in P \cup \Sigma$, and substitutions θ, ρ, the following holds:* $\langle \text{Insert}(U, i, q), \theta \rangle \leqslant 1 \langle \text{Insert}(Z, i, q), \rho \rangle$ *iff* $\langle U, \theta \rangle \leqslant 1 \langle Z, \rho \rangle$ *and* $0 \leqslant i \leqslant \text{Tup}(U)\text{-arity}(q)$.

Proof. $\leqslant 1$ is a partial ordering. (reflexivity) Sub contains the identity substitution ε. (transitivity) $\langle U, \theta \rangle \leqslant 1 \langle Z, \rho \rangle$ and $\langle Z, \rho \rangle \leqslant 1 \langle W, \sigma \rangle$ hence $\langle Z, \rho \rangle = \langle U\theta', \theta\theta' \rangle$ and $\langle W, \sigma \rangle = \langle Z\rho', \rho\rho' \rangle$ hence $\langle W, \sigma \rangle = \langle U\theta'\rho', \theta\theta'\rho' \rangle$. (antisimmetry) Sub does not contain inverses of renamings as adressed in *b*) of Lemma 18.

Finally, for $0 \leqslant i \leqslant \text{Tup}(U)\text{-arity}(q)$, $\langle \text{Insert}(U, i, q), \theta \rangle \leqslant 1 \langle \text{Insert}(Z, i, q), \rho \rangle$ iff τ exists s.t. $\langle \text{Insert}(U, i, q)\tau, \theta\tau \rangle = \langle \text{Insert}(U\tau, i, q), \theta\tau \rangle = \langle \text{Insert}(Z, i, q), \rho \rangle$ iff $\langle U\tau, \theta\tau \rangle = \langle Z, \rho \rangle$ iff $\langle U, \theta \rangle \leqslant 1 \langle Z, \rho \rangle$ ∎

Figure 1 is a picture of $[\Sigma(P, \mathcal{V}) \times \text{Sub},\ \leqslant 1]$. Note that, in general, an open formula corresponds to many points in $[\Sigma(P, \mathcal{V}) \times \text{Sub},\ \leqslant 1]$, depending on the different ways it is obtained. For instance, $(A(fYY))$ occurs many times with different contexts. In particular, $\langle A(fYY)), \varepsilon \rangle \leqslant 1$ $\langle (A(fYY)), \{X \leftarrow Y\} \rangle$ holds but the converse does not.

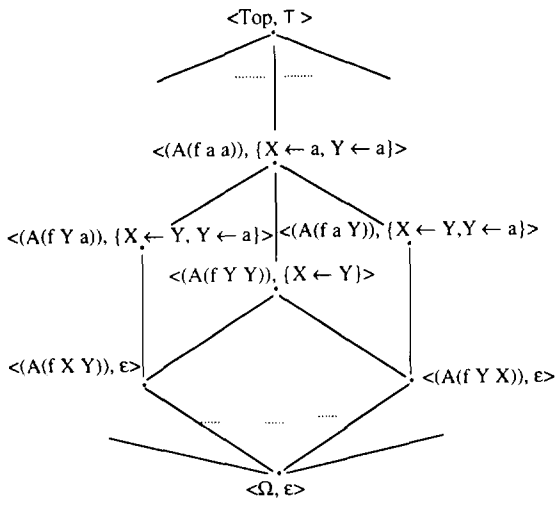

FIG. 1. The partially ordered set $[\Sigma(P, \mathcal{V}) \times \mathrm{Sub}, \leqslant 1]$.

$[\Sigma(P, \mathcal{V}) \times \mathrm{Sub}, \leqslant 1]$ is not a lattice because, for instance, the set $\{\langle U, \sigma\rangle, \langle U, \theta\rangle\}$, with σ and θ as in Eder's example, is bounded from above but has no supremum. However, limiting ourselves to open formulas, we are not interested in the whole $[\Sigma(P, \mathcal{V}) \times \mathrm{Sub}, \leqslant 1]$, but just in those points that represent comparable open formulas and their suprema. Comparable open formulas are pairs $\langle U, \sigma\rangle$ for arbitrary $U \in \Sigma(P, \mathcal{V})$ and the same σ in Sub.

THEOREM 23. *For each finite set of comparable formulas* $C(S) = \{\langle U, \sigma\rangle \mid U \in S\}$ *with* $S \subset \Sigma(P, \mathcal{V})$, $\sigma \in \mathrm{Sub}$, *the supremum in* $[\Sigma(P, \mathcal{V}) \times \mathrm{Sub}, \leqslant 1]$ *exists. Moreover,* $\mathrm{lub}(C(S)) = \langle S\theta, \sigma\theta\rangle$, *where* $\theta = \mathrm{mgu}(S)$. *In particular, if* S *is not unifiable, i.e.,* $\mathrm{mgu}(S) = \top$, *then* $\mathrm{lub}(C(S)) = \langle S\top, \sigma\top\rangle = \langle \mathrm{Top}, \top\rangle$.

Proof. First note that for each set $C(S)$, $\langle S\theta, \sigma\theta\rangle$ is an upper bound since $\theta = \mathrm{mgu}(S)$. Then $S\theta \in \Sigma(P, \mathcal{V})$. In particular, if $\theta = \top$ then $S\theta = \mathrm{Top}$. Let $\langle S\theta', \sigma\theta'\rangle$ be any other upper bound. Then $\theta \leqslant_0 \theta'$ ∎

This proposition shows a strict relationship among $C(S)$, S, $\mathrm{lub}(C(S))$, and $\mathrm{mgu}(S)$. $\mathrm{Lub}(C(S))$ is a pair $\langle U, \sigma\theta\rangle$ and θ has the property of collapsing set S into a single formula, U. Any unifier of S has such property but the formula U is the most general of such formulas and is here called *shrink*. Shrinks are univocally defined in $\Sigma(P, \mathcal{V})$, as well as mgu's are in Sub. The following proposition exploits the relationship between shrinks and mgu's.

PROPOSITION 24. *Let* S *be any finite subset of* $\Sigma(P, \mathcal{V})$ *and* $U = \mathrm{shrink}(S)$, $\theta = \mathrm{mgu}(S)$, *then the following hold:*

(a) $U = S\theta \in \Sigma(P, \mathcal{V})$ *and* $U\theta = U$.

(b) $D(\theta) \subseteq \mathrm{Var}(S)$ *and* $R(\theta) \subseteq \mathrm{Var}(U)$,

(c) *For each* $x \in D(\theta)$, $x\theta = U/\mathrm{Ph}(S, x)$.

(d) *For each* $Z \in S$, *let* ρ *s.t. for each* $x \in \mathcal{V} - \mathrm{Var}(Z)$, $x\rho = x$, *and for each* $x \in \mathrm{Var}(Z)$, *let* $x\rho = U/\mathrm{Ph}(Z, x)$. *Then* ρ *is a restriction of* θ *to* $\mathrm{Var}(Z)$; *i.e.,* $\rho = \theta \upharpoonright (\mathrm{Var}(Z))$.

Proof. (a) By Theorem 20, $\theta = \mathrm{mgu}(S)$ is unique in Sub and idempotent.

(b) If $D(\theta)$ is not included in $\mathrm{Var}(S)$ then $\theta \upharpoonright \mathrm{Var}(S)$ is a unifier, which is smaller than θ. The same holds for $\theta \upharpoonright V$ with $V = \{x \mid \mathrm{Var}(x\theta) \subseteq \mathrm{Var}(U)\}$.

(c) $U/\mathrm{Ph}(S, x) = S\theta/\mathrm{Ph}(S, x) = (S/\mathrm{Ph}(S, x))\theta = x\theta$.

(d) Straightforward from (c) above. ∎

3.2. *Computation of Shrinks*

Shrinks occur in frameworks such as factoring, discussed in Section 4. More recently, (Cline, 1990) presents a method for n-ary linear unification using the computation of some terms. Such terms can also be obtained as shrinks of suitable sets. Given a set S, the computation of the shrink of S is actually obtained in two passes. First, unification is applied, and then, using the resulting mgu, instantiation, as shown in (a) of Proposition 24. This requires the two operations of unification and instantiation, has the cost of such operations, and hence has a lower bound that is linear in the number of symbols in S. The main drawbacks of this approach are (1) the use of unification and instantiation, which makes shrinks dependent on such operations; (2) the two-pass computation, which does not supply the calculus with shrink approximations. Hence, it is not adequate for approximation based calculi, such as the lazy calculus (Friedman and Wise, 1976). We show a direct computation of shrinks, which (i) overcomes the above limitations, (ii) proceeds in one pass, and (iii) behaves as a combination of lazy unification and instantiation. It computes shrink approximations instantiating variables as soon as bindings are computed. The computation obeys the Herbrand's rules for unification via *solved form equation sets* (Lassez *et al.*, 1988; Herbrand, 1930). Such rules are suitably reformulated (i) to deal with set of terms rather than set of equations, (ii) to infer shrinks rather than a set of variable bindings, and (iii) to use a variable selection strategy based on *root classes* (Martelli and Montanari, 1982). This selection strategy is common to the linear algorithms for unification (Paterson and Wegman, 1978; Martelli and Montaneri, 1982), it gains *occur check* for free, and it is at the basis of linear algorithms for shrinks, discussed in Section 3.3.

Let $\aleph$ be the set of non-negative integers. Let S, S', S'' be sets of sequences, let U, T, U_i, and T_i be sequences, let v_i be variables, and let $(f t_1 \cdots t_n)$ be an atom with main symbol f and $t_1 \cdots t_n$ as arguments. Moreover, assume S' to be a non-empty set. We then have

top. $\text{mgi-o}(S' \cup \{\text{Top}\}) = \text{Top}$

bottom. $\text{mgi-o}(S' \cup \{\Omega\}) = \text{mgi-o}(S')$

singleton. $\text{mgi-o}(\{U\}) = U$

merge. $S = \{U, T\} \cup S''$, $U = U1\ v1\ U2$, $T = T1\ v2\ T2$

$v1 \neq v2$, $\text{tup}(U1) = \text{tup}(T1)$, $v1 \leqslant 1\ v2$

$\text{mgi-o}(S) = \text{mgi-o}(S\{v1 \leftarrow v2\})$

var–fun. $S = \{U, T\} \cup S''$, $U = U1\ v1\ U2$, $T = T1(f\,t_1 \cdots t_n)\ T2$

$\text{tup}(U1) = \text{tup}(T1)$, $\text{Paths}(S, v1) \subset \aleph$

$\text{mgi-o}(S) = \text{mgi-o}(S\{v1 \leftarrow (f\,t_1 \cdots t_n)\})$

fun–fun. $S = \{U\} \cup S'$, $U = U1(f\,t_1 \cdots t_n)\ U2$

$F = S'/\text{tup}(U1)$, $F \cap \mathcal{V} = \{\ \}$

$\text{Main}(F) = \{f\}$, $S'' = \text{Remove}(S, \text{tup}(U1))$

$\text{mgi-o}(S) = \text{Insert}(\text{mgi-o}(S''), \text{tup}(U1), f)$

failure. if no other rules apply

$\text{mgi-o}(S) = \text{Top}$

The first three rules are quite evident: *top* considers sets including the Top formula, *bottom* considers sets including the bottom formula, while *singleton* considers sets containing exactly one formula. If the shrink of the set is not Top, *singleton* ultimately applies. The next three rules are a quite plain reformulation of Herbrand's rules of unification, which copes with shrinks and combines unification and instantiation. In particular, *merge* considers the matching of two distinct variables. The binding must preserve the $\leqslant 1$ ordering on variables. *Var–fun* considers the matching of a variable with a non-variable atom. The constraint on $\text{Paths}(S, v1)$ implicitly checks the non-variable atom for the occurrences of the variable (Plaisted, 1984). Moreover, it restricts the rules applicability to root variables, i.e., variables that do not occur as subterms. Hence, the atoms to be reduced further on do not grow after *var–fun*. (Paterson and Wegman, 1978; Martelli and Montanari, 1982) guarantee the existence of such root variables if the set is unifiable. *fun–fun* checks a complete set of corresponding atoms for homogeneity (Paterson and Wegman, 1978) and, using the homeomorphic property of $\leqslant 1$ ordering, removes the main symbols to consider the corresponding arguments. Finally, *failure* returns Top if the set is not a singleton and no other rule applies.

THEOREM 25. *For each finite* $S \subseteq \Sigma(P, \mathcal{V})$, $\text{shrink}(S) = \text{mgi-o}(S)$, i.e., $\text{lub}\{\langle U, \sigma\rangle \mid U \in S\} = \langle \text{mgi-o}(S), \sigma\theta\rangle$ *with* $\theta = \text{mgu}(S)$.

Proof. The rules defined above form a terminating reduction system. We prove first that (1) each rule applies finitely many times, and then (2) a rule applies to each set S resulting in either a sequence or a set having the same shrink of S. Hence the reduction process always stops computing the shrink.

(1) This is obvious for *top*, *bottom*, *singleton*, and *failure*. The rules *merge* and *var–fun* remove all occurrences of a variable at one time. Hence they cannot apply more times than the number V of the distinct variables in the original S. Finally, *fun–fun* removes main symbols from atoms. Atoms may be duplicated by *var–fun*. Atoms do not grow since *var–fun* only replaces root variables, but variable occurrences may exponentially increase. Hence, in addition to the number F of predication/function symbols in the original S, *fun–fun* applies to the duplicated symbols. Such duplications are no more than $O{*}K^V{*}A$, where K and A are the maximum number of variables and of predication/function symbols in the atoms of the original S, and O depends on the multiple occurrences of variables.

(2) This is obvious for the rules *top*, *bottom* and *singleton*. For *fun–fun*, it follows straightforwardly from the homeomorphic property of $\leqslant 1$ ordering. For *failure*, we prove that the rule applies iff S (does not contain Top and) is not unifiable. In fact, *failure* applies iff S contains at least two sequences that differ for at least one of the corresponding atoms. Let u, t be such atoms. The atom u must be a non-variable atom (otherwise *merge* applies). If t is a variable and *var–fun* does not apply then variables in S cannot be partially ordered and unification generates an occur check (Paterson and Wegman, 1978). Otherwise, t must be a non-variable atom and, since *fun–fun* cannot apply, a mismatch on the main symbols occurs. Finally, consider *merge* and *var–fun*. Both replace S by $S\{v1 \leftarrow t\}$, where t is a variable and a non-variable atom, respectively. Let θ be $\text{mgu}(S)$. Since $\{v1 \leftarrow t\}$ is the smallest substitution unifying $v1$ and t, $\{v1 \leftarrow t\} \leqslant_o \theta$. Then, for an arbitrary substitution σ and for each sequence U in S, we have

$$\langle U, \sigma\rangle \leqslant 1\ \langle U\{v1 \leftarrow t\}, \sigma\{v1 \leftarrow t\}\rangle \leqslant 1\ \langle U\theta, \sigma\theta\rangle = \langle \text{Shrink}(S), \sigma\theta\rangle;\ \text{thus}$$
$$\langle \text{Shrink}(S), \sigma\theta\rangle = \text{lub}\langle S, \sigma\rangle \leqslant 1\ \text{lub}\langle S\{v1 \leftarrow t\}, \sigma\{v1 \leftarrow t\}\rangle \leqslant 1\ \langle U\theta, \sigma\theta\rangle.$$ ∎

EXAMPLE. Let $S = \{(f\,X\,Y), Z, (f\,Y\,X), (f\,W\,a)\}$ with $\{X \leqslant 1\ Y \leqslant 1\ Z \leqslant 1\ W\}$. Then we have

$$\frac{S = \{(f\,X\,Y), Z, (f\,Y\,X), (f\,W\,a)\},\ U = Z,\ T = (f\,X\,Y)}{\text{mgi-o}(S) = \text{mgi-o}(S\{Z \leftarrow f\,X\,Y\}) = \text{mgi-o}(S1)}\ \textit{var–fun}$$

$$\frac{S1 = \{(f\,X\,Y), (f\,Y\,X), (f\,W\,a)\},\ U = (f\,X\,Y),\ S1'' = \text{Remove}(S1, 0) = S2}{\text{mgi-o}(S1) = \text{Insert}(\text{mgi-o}(S2), 0, f)}\ \textit{fun–fun}$$

$$S2 = \{X\,Y,\ Y\,X\} \cup \{W\,a\},\ U = X\,Y,\ T = Y\,X,\ v1 = X,\ v2 = Y,\ X \leqslant 1\ Y$$

$$\frac{S2'' = S2\{X \leftarrow Y\} = S3}{\text{mgi-o}(S2) = \text{mgi-o}(S3)}\ \textit{merge}$$

$$\frac{S3 = \{Y\ Y,\ W\ a\},\ U = Y\ Y,\ T = W\ a,\ v1 = Y,\ v2 = W \qquad S3'' = S3\{Y \leftarrow W\} = S4}{\text{mgi-o}(S3) = \text{mgi-o}(S4)}\ \textit{merge}$$

$$\frac{S4 = \{W\ W,\ W\ a\},\ U = W\ W,\ T = W\ a,\ v1 = W}{\text{mgi-o}(S4) = \text{mgi-o}(S4\{W \leftarrow a\}) = \text{mgi-o}(\{a\ a\}) = a\ a}\ \textit{var-fun}$$

singleton

Hence mgi-o(S) = (f a a). (See Fig. 2.)

The computation of mgi-o(S) proceeds on majorizing sequences of S as in Fig. 2. It results in a sequence of approximations to the shrink of S. Approximations can be used in exploiting the form of the possible solutions or in stopping the computation when unexpected approximations are generated or some constraints are violated. In the example above, after two reductions, we obtain the approximation Insert(mgi-o($S2$), 0, f). This says that the shrink either has the form (f t_1 t_2) or is Top. If we are looking for an atom whose main symbol is $g \neq f$, we can immediately stop the computation.

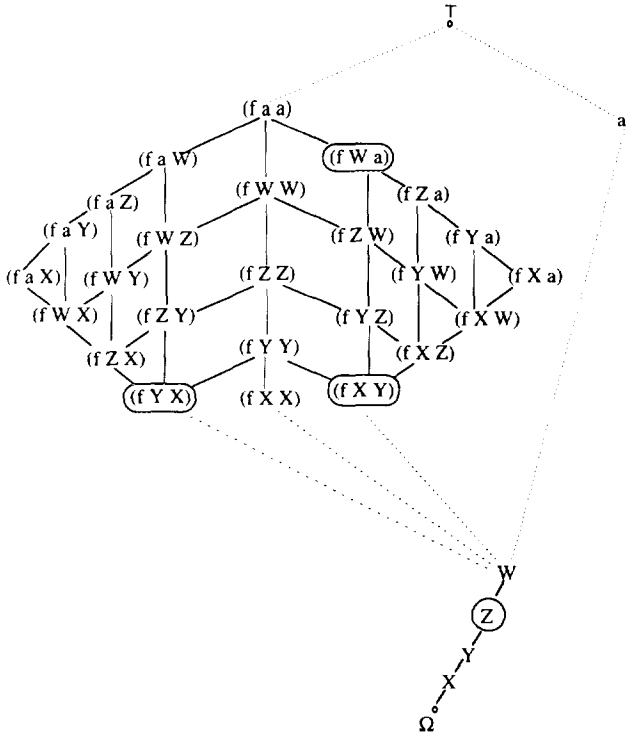

FIG. 2. Computation of the shrink of the set $S = \{(f\ X\ Y),\ Z,\ (f\ Y\ X),\ (f\ W\ a)\}$.

3.3. *A Linear Algorithm for* Mgi-o

Though terminating and, because of uniqueness of shrink, confluent, the system mgi-o is not deterministic. Given a set S of formulas, finitely many different reduction sequences exist. Some of them may be of exponential length with the number $V + F$ of variables and predication/function symbols occurring in S. As discussed in the termination part of the proof of Theorem 25, length has upper bound $V + F + O{*}K^{V}{*}A$. In fact, though atoms do not grow, variable occurrences may exponentially increase. Correspondingly, *var–fun* may generate exponential duplications of the same atom. Consider, for instance, the set $S = \{X1\ X2\ X3,\ (f\ X0\ X0)\ (f\ X1\ X1)\ (f\ X2\ X2)\}$. Only $X3$ can be initially selected for the rule *var–fun*, resulting in the following reductions:

$$\frac{S = \{X1\ X2\ X3,\ (f\ X0\ X0)\ (f\ X1\ X1)\ (f\ X2\ X2)\} \qquad v1 = X3}{\text{mgi-o}(S) = \text{mgi-o}(S\{X3 \leftarrow (f\ X2\ X2)\}) = \text{mgi-o}(S1)}\ \textit{var-fun}$$

$$\frac{S1 = \{X1\ X2\ (f\ X2\ X2),\ (f\ X0\ X0)\ (f\ X1\ X1)\ (f\ X2\ X2)\} \qquad \text{tup}(U1) = 2}{\text{mgi-o}(S1) = \text{Insert}(\text{mgi-o}(S2), f, 2)}\ \textit{fun-fun}$$

$$S2 = \{X1\ X2\ X2\ X2,\ (f\ X0\ X0)\ (f\ X1\ X1)\ X2\ X2\}$$

Hence, *var–fun* removes the variable $X3$ and *fun–fun* removes the main symbol f, but now variable $X2$ occurs twice more. The replacement of $X2$ by an atom yields the rule *fun–fun* to consider the last two occurrences of $X2$ as standing for atoms to be reduced.

A reduction sequence of linear length with $V + F$ may be always derived under a suitable control on the application of the rules. The control applies the rule *merge* first. When the rule *var–fun* applies, atom duplications are tagged blocked. For instance, we proceed with the reductions of the example above. Variable $X2$ is selected:

$$\frac{v1 = X2}{\text{mgi-o}(S2) = \text{mgi-o}(S2\{X2 \leftarrow (f\ X1\ X1)_b\}) = \text{mgi-o}(S3)}\ \textit{var-fun}$$

$$S3 = \{X1\ (f\ X1\ X1)_b\ (f\ X1\ X1)_b\ (f\ X1\ X1)_b,\ (f\ X0\ X0)(f\ X1\ X1)(f\ X1\ X1)_b\ (f\ X1\ X1)_b\}$$

The blocking mechanism prevents the rule *fun–fun* from applying to sets containing only duplications of the same atom. In the above example the reductions complete as follows:

$$\frac{\text{tup}(U1) = 1}{\text{mgi-o}(S2) = \text{Insert}(\text{mgi-o}(S3), f, 1)}\ \textit{fun-fun}$$

$$S3 = \{X1\ X1\ X1\ (f\ X1\ X1)_b\ (f\ X1\ X1)_b,\ (f\ X0\ X0)\ X1\ X1\ (f\ X1\ X1)_b\ (f\ X1\ X1)_b\}$$

$v1 = X1$

var–fun

$\text{mgi-o}(S3) = \text{mgi-o}(S3\{X1 \leftarrow (f\,X0\,X0)_b\}) = \text{mgi-o}(S4)$

$S4 = \{(f\,X0\,X0)(f\,X0\,X0)_b\,(f\,X0\,X0)_b\,(f(f\,X0\,X0)$
$(f\,X0\,X0))_b\,(f(f\,X0\,X0)(f\,X0\,X0))_b,$
$(f\,X0\,X0)_b\,(f\,X0\,X0)_b\,(f\,X0\,X0)_b\,(f(f\,X0\,X0)(f\,X0\,X0))_b$
$(f(f\,X0\,X0)(f\,X0\,X0))_b\}$

$\text{tup}(U1) = 0$

fun–fun

$\text{mgi-o}(S4) = \text{Insert}(\text{mgi-o}(S5), f, 0)$

$S5 = \{X0\,X0\,(f\,X0\,X0)(f\,X0\,X0)(f(f\,X0\,X0)(f\,X0\,X0))$
$(f(f\,X0\,X0)(f\,X0\,X0))\}$

$\text{mgi-o}(S5) = X0\,X0\,(f\,X0\,X0)(f\,X0\,X0)(f(f\,X0\,X0)$
$(f\,X0\,X0))(f(f\,X0\,X0)(f\,X0\,X0))$

singleton

In this way, *fun–fun* always removes at least one non-duplicated symbol. Hence a linear algorithm for mgi-o can be obtained by superimposing such a control on rules application.

4. FACTORING

Factoring has been introduced for several purposes (Chang and Lee, 1974). In the resolution process the need for it arises from the definition of criteria for reducing the generation of resolvents and controlling subsumptions and tautologies. Let $U = u_1 \vee \cdots \vee u_n$ be a clause in disjunctive form with n literals u_i. We say that u is a *factor* of $z_1 \cdots z_k$, where $\{z_1, ..., z_k\} \subseteq \{u_1, ..., u_n\}$, if:

(i) $\{z_1, ..., z_k\}$ is unfiable with $\text{mgu}\{z_1, ..., z_k\} = \theta (\neq \top)$, and

(ii) $u = \{z_1, ..., z_k\}\theta$.

The formula resulting from the instantiation of U with θ is called the *factoring* of U w.r.t. $\{z_1, ..., z_k\}$, i.e., $U\theta$. Moreover, the clause $U(u)$, obtained from $U\theta$ by removing all but one occurrences of u, is called the *factorized* formula of U. Factorings (and factorized formulas) of U are always subsumed from U.

Given a clause $U = u_1 \vee \cdots \vee u_n$ we assume $[U]$ to be the sequence of SOAF such that $[U] = [u_1] \cdots [u_n]$ with $[u_i] = u_i$ for each positive literal, and $[\text{not}(f\,t_1 \cdots t_n)] = (\sim f\,t_1 \cdots t_n)$ for each negative literal. The computation of factors and factorings of clauses is then expressed using mgi-o in the following way. Given a clause U, let $Z = [U]$ be the sequence of $\Sigma(P, \mathcal{V})$ denoting U as above, and let I be the set of paths in Z such that $Z/I = \{[z_1], ..., [z_k]\}$. Then

$$\text{mgi-o}(Z/I) = u.$$

Moreover, let $Z(i) = Z\{(I - \{i\}) \leftarrow Z/\{i\}\}$, and $Z^* = \{Z(i) \mid i \in I\}$. Then

$$\text{mgi-o}(Z^*) = U\theta$$

and the factorized formula of U is

$$\text{Remove}(\text{mgi-o}(Z^*), I - \{i\}) = U(u)$$

for an arbitrary i in I.

EXAMPLE. Let $U = (A\,X) \vee (A\,0) \vee (B\,X)$, where $\{(A\,X), (A\,0)\}$ is unifiable. Then for $[U] = (A\,X)(A\,0)(BX)$ and $I = \{0, 1\}$ we have $Z/I = \{(A\,X), (A\,0)\}$; thus the factor is $\text{mgi-o}(Z/I) = (A\,0)$. Moreover, $Z^* = \{(A\,X)(A\,X)(B\,X), (A\,0)(A\,0)(B\,X)\}$; thus

$$\text{mgi-o}(Z^*) = U\theta = (A\,0)(A\,0)(B\,0)$$

and finally the factorized formula $U = (A\,0)(B\,0)$.

Using mgi-o, the computation of the factoring requires the computation of Z^*. This introduces a duplication of subformulas in U. Even though this does not significantly affect time efficiency (reductions are the same as in the computation of the factor u), space efficiency is useless decreased by the length of formulas in Z^*. We can avoid it and the use of Z^* simply extending the expressiveness of mgi-o (Bellia and Occhiuto, 1993).

5. AN ALGEBRA FOR CLOSED FORMULAS: SCAF

(Reynolds, 1970) already introduced an algebra for atomic closed formulas. Among other features, the algebra provides for the gci($\sqcap$) and lcg($\sqcup$) operators, which compute the lub and the glb respectively, in the lattice with instance as the partial ordering for formulas. Reynolds also introduces the anti-unification algorithm to express lcg, while he uses unification to compute gci. However it is not possible to compute the gci for any pair of formulas by the simple application of the unification algorithm. In general, we need to consider suitable variants of the formulas. The reason for variants inests upon the properties of the algebraic structure of closed formulas. We briefly revisit them and, at the same time, extend the structure of the formulas to those syntactically considered in $\Sigma(P, \mathcal{V})$. Reynolds used the algebra as the formal basis for refutation in transformational systems (Reynolds, 1970), a restricted form of Horn clause theories. Recently, gci has been used as a semantic operator, called mgi, for the fixpoint semantics proposed by (Falaschi *et al.*, 1989), which models the operational behavior of non-ground atomic formulas of logic programs.

Top and Ω, previously introduced, are the universal and the null formula in (Reynolds, 1970), respectively. Each sequence $u_1 \cdots u_n \in \Sigma(P, \mathcal{V})$ is a closed formula where variables are all (for instance, universally) quantified.

DEFINITION 26. Let $\leqslant 2$ be the instance preorder defined as follows:

- $\Omega \leqslant 2\ U, \forall U \in \Sigma(P, \mathcal{V})$
- $U \leqslant 2$ Top, $\forall U \in \Sigma(P, \mathcal{V})$
- $U \leqslant 2\ U\theta, \forall U \in \Sigma(P, \mathcal{V})$ and $\theta \in$ Subst.

$\leqslant 2$ defines an equivalence relation on formulas: $U \sim Z \Leftrightarrow U \leqslant 2\ Z$ and $Z \leqslant 2\ U$. This relation coincides with the variant relation for closed formulas, i.e., $U \sim Z \Leftrightarrow U \in$ Variant$(Z) = \{Z\theta \mid \theta \in$ permutations$\}$.

PROPOSITION 27. *$\leqslant 2$ is homeomorphic in $\Sigma(P, \mathcal{V})$, i.e., for each U and Z of the same tupling and $q \in P \cup \Sigma$ of the correct arity,* Insert$(U, i, q) \leqslant 2$ Insert$(Z, i, q) \Leftrightarrow U \leqslant 2\ Z$ *and* $0 \leqslant i \leqslant$ Tup(U)-arity(q).

THEOREM 28. *$[\Sigma(P, \mathcal{V})/\sim, \leqslant 2]$ is a lattice.*

Proof. Let $U, Z \in \Sigma(P, \mathcal{V})$. Then we prove that

- $\exists U \sqcap Z \in \Sigma(P, \mathcal{V})$, s.t.

  $U \leqslant 2\ U \sqcap Z$, $Z \leqslant 2\ U \sqcap Z$, and $\forall W \in \Sigma(P, \mathcal{V})$,

  $U \leqslant 2\ W$, $Z \leqslant 2\ W \Rightarrow U \sqcap Z \leqslant 2\ W$.

- $\exists U \sqcup Z \in \Sigma(P, \mathcal{V})$, s.t.

  $U \sqcup Z \leqslant 2\ U$, $U \sqcup Z \leqslant 2\ Z$, and $\forall W \in \Sigma(P, \mathcal{V})$,

  $W \leqslant 2\ U$, $W \leqslant 2\ Z \Rightarrow W \leqslant 2\ U \sqcup Z$.

When tup$(U) \neq$ tup(Z), $U \sqcap Z =$ Top and $U \sqcup Z = \Omega$. Otherwise, let $A^q = \{$Insert$(W, 0, q) \mid q \notin \Sigma \cup P$, arity$(q)$ = Tup(U), $W \in \Sigma(P, \mathcal{V})$, Tup$(W)$ = Tup$(U)\}$. Then $A^q \subseteq \Sigma'(\mathcal{V})$ for $\Sigma' = \Sigma \cup P \cup \{q\}$ and Insert$(U, 0, q) \sqcap$ Insert$(Z, 0, q)$ and Insert$(U, 0, q) \sqcup$ Insert$(Z, 0, q)$ exist in A^q because of Theorem 1 and Theorem 2 in (Reynolds, 1970). Moreover, Insert$(U, 0, q) \sqcap$ Insert$(Z, 0, q) =$ Insert$(U \sqcap Z, 0, q)$ and Insert$(U, 0, q) \sqcup$ Insert$(Z, 0, q) =$ Insert$(U \sqcup Z, 0, q)$ because of Proposition 27. ∎

We could easily extend Huet's arguments (Huet, 1980) to show that $\leqslant 2$ is such that each formula (except Top) in $\Sigma(P, \mathcal{V})$ has finitely many predecessors. Hence infinite sets of formulas in $\Sigma(P, \mathcal{V})$ are bounded from above only by Top, hence $[\Sigma(P, \mathcal{V})/\sim, \leqslant 2]$ is also a complete lattice.

Given a set S, the computation of the lub can proceed through the computation of the mgu of S' and then the instantiation of S' with such a unifier. This works correctly, provided that S' is a renaming of S having no colliding variables among distinct formulas. As for shrinks, a drawback of this approach is that no approximations are supplied. Alternatively, we can use the rules for mgi-o and apply them to S'. This works correctly since, having no colliding variables among distinct formulas, S' has lub(S') = shrink(S'). However, in this approach the renaming S' must be completely computed independently of the approximation we are looking for. A solution that allows a lazy computation of the renaming is to convert the rules for the computation of mgi-o into rules that, in addition, compute a variable renaming every time a non-global variable is considered.

Let $\aleph$ be the set of non-negative integers. Let S, S', S'' be sets of sequences, U, T, U_i, and T_i sequences, v and v_i variables, and $(f\, t_1 \cdots t_n)$ an atom with main symbol f and $t_1 \cdots t_n$ as arguments. Moreover, assume S' to be a non-empty set.

top.	mgi-c$(S' \cup \{\text{Top}\}) =$ Top
bottom.	mgi-c$(S' \cup \{\Omega\}) =$ mgi-c(S')
singleton.	mgi-c$(\{U\}) = U$
merge.	$S = \{U, T\} \cup S''$, $U = U1\ v1\ U2$, $T = T1\ v2\ T2$ $v1 \neq v2$, tup$(U1) =$ tup$(T1)$, Global$(v1, S)$ if Global$(v2, S)$ then $S' = S\{v2 \leftarrow v1\}$ else $S' = S'' \cup \{U, T\{v2 \leftarrow v1\}\}$
	mgi-c$(S) =$ mgi-c(S')
rename.	$S = \{U\} \cup S'$, $U = U1\ v1\ U2$
	mgi-c$(S) =$ mgi-c$(S' \cup \{U\{v \leftarrow v1, v1 \leftarrow v\}\})$
var–fun.	$S = \{U, T\} \cup S''$, $U = U1\ v1\ U2$, $T = T1\ (f\, t_1 \cdots t_n)\ T2$ tup$(U1) =$ tup$(T1)$, Paths$(S, v1) \subset \aleph$, Global$(v1, S)$
	mgi-c$(S) =$ mgi-c$(S\{v1 \leftarrow$ Ren$(f\, t_1 \cdots t_n)\})$
fun–fun.	$S = \{U\} \cup S'$, $U = U1\ (f\, t_1 \cdots t_h)\ U2$ $F = S'/$tup$(U1)$, $F \cap \mathcal{V} = \{\ \}$ Main$(F) = \{f\}$, $S'' =$ Remove$(S,$ tup$(U1))$
	mgi-c$(S) =$ Insert(mgi-c(S''), tup$(U1), f)$
failure.	if no other rule applies
	mgi-c$(S) =$ Top

Rules *top*, *bottom*, *singleton*, *fun–fun*, and *failure* are the same as for shrinks. *merge* and *var–fun* use a predicate Global, which applies to a variable and a set of sequences. Global(v, S) is true if unification would consider each occurrence of v in S as a different occurrence of the same variable. That is, unification binds all such occurrences to the same term. Formally, let $R(v)$ be the minimum equivalence relation on sequences of S, such that $U\ R(v)\ T$ if either Paths$(U, v) = \{\ \}$ or Paths$(T, v) = \{\ \}$ or Paths$(U, v) \cap$ Paths$(T, v) \neq \{\ \}$. Then Global(v, S) iff for each U, T in S, $U\ R(v)\ T$. As an example, consider the set $\{(A\ X)(B\ Y\ Z), (A\ (f\ Y\ Z))(B\ 0\ Z)\}$. Variables X and Z are global. In particular, Z occurs in different sequences but one of its occurrences is the same in the two sequences; hence we can consider the Z in the first sequence and the Z in the

second sequence as the same variable. The same does not apply to Y. The Y in the first sequence and the Y in the second sequence cannot be considered the same, hence Y is not global. *merge* applies to at least one global variable $v1$ and replaces $v2$ only in T if $v2$ is not global in S, or in the whole S if $v2$ is global. *var–fun* also applies to a global variable. But to avoid unexpected variable collisions, the binding is renamed using operator Ren. Ren applies to an atom and renames the variables in it with fresh variables, i.e., variables not occurring in S. Finally, the new rule *rename* is introduced. Using permutations, the rule renames variables in order to eliminate non-global variables.

EXAMPLE. Let $S = \{(f\,X\,Y), Z, (f\,Y\,X), (f\,W\,a)\}$:

$$\frac{S = \{(f\,X\,Y), Z, (f\,Y\,X), (f\,W\,a)\},\ U = Z,\ T = (f\,X\,Y)}{\text{mgi-c}(S) = \text{mgi-c}(S\{Z \leftarrow \text{Ren}(f\,X\,Y)\}) = \text{mgi-c}(S1)}\ \textit{var–fun}$$

$$S1 = \{(f\,X\,Y)\} \cup S1',\ S1' = \{(f\,W1\,W2), (f\,Y\,X), (f\,W\,a)\},\ U = (f\,X\,Y)$$

$$\frac{S1'' = \text{Remove}(S, 0) = S2}{\text{mgi-c}(S1) = \text{Insert}(\text{mgi-c}(S2), 0, f)}\ \textit{fun–fun}$$

$$S2 = \{Y\,X\} \cup S2',\ S2' = \{X\,Y, W1\,W2, W\,a\},\ U = Y\,X,\ v1 = X,\ v = Y$$

$$\frac{}{\text{mgi-c}(S2) = \text{mgi-c}(S2' \cup \{Y\,X\{X \leftarrow Y, Y \leftarrow X\}\}) = \text{mgi-c}(S3)}\ \textit{rename}$$

$$S3 = \{X\,Y, W1\,W2\} \cup \{W\,a\},\ U = X\,Y,\ T = W1\,W2,\ v1 = X,\ v2 = W1$$

$$\frac{\text{Global}(X, S3), \text{Global}(W1, S3)}{\text{mgi-c}(S3) = \text{mgi-c}(S3\{W1 \leftarrow X\}) = \text{mgi-c}(S4)}\ \textit{merge}$$

$$S4 = \{X\,Y, X\,W2\} \cup \{W\,a\},\ U = X\,Y,\ T = X\,W2,\ v1 = Y,\ v2 = W2$$

$$\frac{\text{Global}(Y, S4), \text{Global}(W2, S4)}{\text{mgi-c}(S4) = \text{mgi-c}(S4\{W2 \leftarrow Y\}) = \text{mgi-c}(S5)}\ \textit{merge}$$

$$S5 = \{X\,Y, W\,a\},\ U = W\,a,\ T = X\,Y,\ v1 = W,\ v2 = X$$

$$\frac{\text{Global}(W, S5), \text{Global}(X, S5)}{\text{mgi-c}(S5) = \text{mgi-c}(S5\{X \leftarrow W\}) = \text{mgi-c}(S6)}\ \textit{merge}$$

$$S6 = \{W\,Y, W\,a\},\ U = W\,Y,\ T = W\,a$$

$$\frac{}{\text{mgi-c}(S6) = \text{mgi-c}(S6\{Y \leftarrow a\}) = W\,a \qquad \textit{singleton}}\ \textit{var–fun}$$

Hence mgi-c(S) = (f $W1$ a). (See Fig. 3.)

THEOREM 29. *For each finite* $S \subseteq \Sigma(P, \mathcal{V})$, mgi-c($S$) = lub($S$).

Proof. In this case the reduction process is not terminating because of the rule *rename*, which can apply the same permutation infinitely many times. We show that if S is unfiable then a finite reduction exists, which computes lub(S).

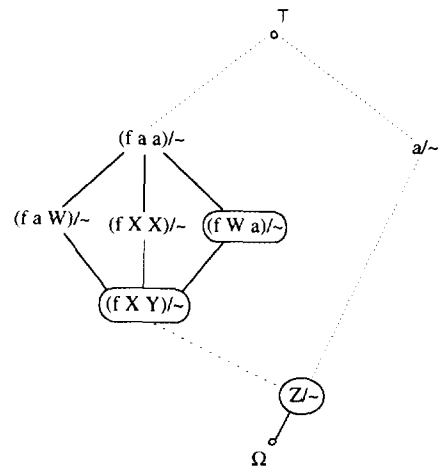

FIG. 3. Computation of the lub of the set $S = \{(f\,X\,Y), Z, (f\,Y\,X), (f\,W\,a)\}$.

(a) Suppose that no colliding variables exist in S, that is, for each U, T in S, Var(U) ∩ Var(T) = { }. Hence, lub(S) = shrink(S) = mgi-o(S) since Theorem 25. In this case all variables in S are global; hence *merge* and *var–fun* apply exactly as the corresponding rules of mgi-o. The renaming of the binding in *var–fun* is useless but not wrong since corresponding variables will be merged eventually. In this case we do not need the rule *rename* and mgi-c(S) = mgi-o(S).

(b) If S contains colliding variables, that is, U and T exist in S such that Var(U) ∩ Var(T) ≠ { }, then we can apply *rename* to U choosing v as a fresh variable. Hence in a finite number of reductions with the rule *rename* (one for each colliding variable) we obtain a set $S' \sim S$, without colliding variables. Thus, we have lub(S) = lub(S') ∎

A renaming strategy can be used to guarantee termination. The strategy applies using only fresh variables and effectively renaming only not already renamed variables. In this way *rename* applies at most V times, where V is the number of variables occurring in S. Obviously, fresh variables are global; thus checks for global variables actually become checks for already renamed variables. This simplifies the predicate global and leads to a linear algorithm using the same blocking mechanism to control rule application in mgi-o.

Briefly we investigate the relations among mgi-c, weak and ordinary unification.

PROPOSITION 30. *Let* $S = \{T_1, ..., T_n\} \subset \Sigma(P, \mathcal{V})$ *and* $T \in \Sigma(P, \mathcal{V})$ *such that* $T \sim$ mgi-c(S). *Then the following propositions hold.*

(a) *Let* $\theta = \{\theta_1, ..., \theta_n\}$ *with* θ_i *be such that* $x\theta_i = T/Ph(T_i, x)$ *for* $x \in \mathrm{Var}(T_i)$ *and,* $x\theta_i = x$ *otherwise. Then* θ *is a weak unifier of* S.

(b) *For each* $i \in [1, n]$, *let* ρ_i *be a renaming on* $\cup_j(\mathrm{Var}(T_i) \cap \mathrm{Var}(T_j))$ *for* $j \in [1, n]$ *and* $j \neq i$, *such that for each* $1 \leqslant i < j \leqslant n$, $\mathrm{Var}(T_i\rho_i) \cap \mathrm{Var}(T_j\rho_j) = \{\ \}$, *and for each* $x \in \mathrm{Var}(T)$, ρ_i *exists such that* $y\rho_i = x$ *and* $T/Ph(T_i, y) = x$.

With θ_i *in* (a) *and* ρ_i *above, let* μ *exist such that* $x\mu = x(\rho_i^-)\theta_i$ *for* $x \in (D(\theta_i) - D(\rho_i)) \cup R(\rho_i)$ *and* $i \in [1, n]$, *and* $x\mu = x$ *otherwise.*

Then μ *is the most general unifier of* $\{T_1\rho_1, ..., T_n\rho_n\}$. *Moreover, we have* $T_1\rho_1\mu = \cdots = T_n\rho_n\mu = T$ *and* μ *is idempotent.*

Proof. (a) $T_1\theta_1 = \cdots = T_n\theta_n = T$. For (b), first note that $(R(\rho_1) \cup \cdots \cup R(\rho_n)) \cap \mathrm{Var}(T) = \mathrm{Var}(T)$. Then μ is an idempotent substitution with $D(\mu) = \cup_i (R(\rho_i) \cup (D(\theta_i) - D(\rho_i))) - \mathrm{Var}(T)$ *for* $i \in [1, n]$ and $R(\mu) = \mathrm{Var}(T)$. Moreover, $T_i\rho_i\mu = T_i\rho_i(\rho_i^-)\theta_i = T_i\rho_i^-\theta_i$ by Lemma 10, and $T_i\rho_i^-\theta_i = T_i\theta_i = T$. Then μ is a unifier and also, a most general unifier since $T_i\theta_i$ is $\mathrm{lub}\{T_1\rho_1, ..., T_n\rho_n\}$ ∎

6. RESOLUTION

Robinson introduced unification as the basic operation to express and then, compute resolvents of clauses. A clause (in disjunctive form) is a universally quantified disjunction of literals, i.e., positive and negative atomic formulas: $C = \forall X,\ c_1 \vee \cdots \vee c_n$, where $c_i = a_i$ or $c_i = (\mathrm{not}\ a_i)$ for some atomic formula a_i. To represent C, Robinson introduced the quad notation (Robinson, 1979). When convenient, he regards C as the set of its literals and writes: $C = C_P \cup C_N$ where C_P and C_N are respectively the subset of positive and the subset of negative atomic formulas. We will use this set notation. In addition, we introduce operators p and n to select the C_P and C_N of a clause C, i.e., $p(C) = C_P$, $n(C) = C_N$.

According to (Robinson, 1965, 1979), the general form of (binary) resolvents is the following. Let C_1 and C_2 be two (not necessarily distinct) clauses in a finite set S of clauses, such that

(a) C_1 and C_2 have no colliding variables, i.e., $\mathrm{Var}(C_1) \cap \mathrm{Var}(C_2) = \{\ \}$. Otherwise, we consider a variant of them by renaming the variables, for instance, in C_1 away from $\mathrm{Var}(C_2)$, i.e., $C_1' = C_1\rho_1$;

(b) F is a non-empty subset of $n(C_1)$, and G is a non-empty subset of $p(C_2)$; i.e., $n(C_1) = E \cup F \neq \{\ \}$ and $p(C_2) = H \cup G \neq \{\ \}$ for possibly empty E and H. Moreover, $E \cap F = H \cap G = \{\ \}$;

(c) F and G are unifiable, i.e., $\mathrm{mgu}(\{F, G\}) = \theta \neq \top$. Then:

(d) $R = (p(C_1) \cup (p(C_2) - G) \cup (n(C_1) - F) \cup n(C_2))\,\theta$. R is a resolvent of C_1 and C_2.

Note the crucial role of variants and unification in this definition. In its original formulation F and G are (arbitrarily) chosen among the unifiable subset of $n(C_1)$ and $p(C_2)$, respectively, which are unifiable in between. When factoring is used, F and G are singleton sets, i.e., $F = \{\mathrm{not}\ f\}$ and $G = \{g\}$. When factoring is not used, F and G must be suitably chosen as the maximum of such unifiable subsets. This avoids useless and expensive computations of subsumed Rs. It is convenient, without loss of generality, to consider C_1 and C_2 already separately factored, and F and G singleton sets (or simply literals). In Section 4 we have discussed factoring for open formulas, it applies to the subformulas of a clause considering them as open formulas.

Using mgi-c as first class operation, we give a new formulation of resolution. The formulation stays within the structure of formulas. Hence, it does not resort to the structure of substitutions and uses neither unification nor instantiation. Let $[C_1]$ and $[C_2]$ be sequences denoting the clauses C_1 and C_2 as in Section 4, I_1 and I_2 be the sets of paths such that $C_1/I_1 = F = \{\mathrm{not}\ f\}$ and $C_2/I_2 = G = \{g\}$. Moreover, let V_1 and V_2 be two sequences of length $\mathrm{tup}([C_1])$ and $\mathrm{tup}([C_2])$ respectively, such that:

$V_1/I_1 = \{[\mathrm{not}\ g]\}$, and
$V_1/(\{0 \leqslant i < \mathrm{tup}(V_1)\} - I_1) \subseteq X \subset \mathcal{V} - \mathrm{Var}(C_2) = \{\ \}$;

$V_2/I_2 = \{[f]\}$, and
$V_2/(\{0 \leqslant i < \mathrm{tup}(V_2)\} - I_2) \subseteq Y \subset \mathcal{V} - \mathrm{Var}(C_1) = \{\ \}$;

V_1 (resp. V_2) has the complementary literal in G (resp. F) at the occurrences in I_1 (resp. I_2) and is completed with distinct variables from the set X (resp. Y). Then we define

$$Z = \text{mgi-c}\{(V_1\ [C_2]), ([C_1]\ V_2)\},$$

where $(V\ C)$ is denoting the sequence resulting from the concatenation of sequence V with C. Theorem 31 shows that Z, in which atoms at $I_1 \cup \{\mathrm{tup}(C_1) + i \mid i \in I_2\}$ are removed, is the resolvent R.

EXAMPLE. Let $C_1 = (A\ X\ Y) \vee \mathrm{not}\ (B\ X) \vee (B\ Y)$ and $C_2 = (B\ 0)$ be two clauses. Let $F = \{\mathrm{not}\ (B\ X)\} = C_1/\{1\}$ and $G = \{(B\ 0)\} = C_2/\{0\}$ be selected for the resolution. Then we consider the sequences

$$[C_1] = (A\ X\ Y)(\sim B\ X)(B\ Y) \quad [C_2] = (B\ 0)$$

and we have

$$V_1 = W_1(\sim B\ 0)\ W_2 \quad V_2 = (B\ X),$$

where $W1$ and $W2$ are arbitrary distinct variables in $X \subset \mathcal{V} - \mathrm{Var}(C_2)$. Hence

$$Z = \text{mgi-c}\{W_1(\sim B\,0)\; W_2(B\,0), (A\; X\; Y)(\sim B\; X)\; (B\; Y)(B\; X)\}$$
$$= (A\; 0\; Y)(\sim B\,0)(B\; Y)(B\,0)$$

The resolvent is $R = \mathrm{Remove}(Z, \{1, 3\}) = (A\; 0\; Y)(B\; Y)$.

THEOREM 31. *Let R, θ, and Z be as above. Then*

- *Z is (denoting) a tautology;*
- $Z \sim [R \cup F\theta \cup G\theta]$.

Proof. For the first part: $(V1\ [C_2])$ and $([C_1]\ V_2)$ are tautologies and ordering $\leqslant 2$ preserves tautologies. For the second part: For each $i \in [0, \mathrm{tup}(V_1) + \mathrm{tup}(V_2) - 1]$, $(V_1\ [C_2])/i$ and $([C_1]\ V_2)/i$ differ because if either one is a variable, not occurring in the rest of the sequence, or the first one is $[f]$ and the second one is $[g]$ (in the case $i \in I_1$), or symmetrically, first one is [not f] and second one is [not g] (in the case $i \in I_2$). By construction of $(V_1\ [C_2])$, $([C_1]\ V_2)$, if f and g are unifiable with mgu θ then, assuming $X \cap Y = \{\ \}$, $(V_1\ [C_2])$, $([C_1]\ V_2)$ is also unifiable with mgu θ. Hence, mgi-c$\{(V_1\ [C_2]), ([C_1]\ V_2)\}$ exists because of Proposition 30. Let θ_1, θ_2 be weak unifiers of $(V_1\ [C_2])$, $([C_1]\ V_2)$ as in (b) of Proposition 30. Then $\mu = \theta_1\theta_2 = \theta_2\theta_1$ is mgu of $\{(V_1\ [C_2]), ([C_1]\ V_2)\}$ since $(V_1\ [C_2])$, $([C_1]\ V_2)$ have no colliding variables and, for $\rho_1 = \rho_2 = \varepsilon$, μ is the mgu in (b) of Proposition 30. Hence, $\theta \sim \mu$ and mgi-c$\{(V_1\ [C_2]), ([C_1]\ V_2)\} = (V_1\ [C_2])\,\mu \sim (V_1\ [C_2])\,\theta = [R \cup F\theta \cup G\theta]$. ∎

Computing Z we start from a pair of clauses C_1, C_2 with no colliding variables as required from premise (a) of Robinson's definition. During a deduction this assumption is hard since the same clause may be needed several times. Then it requires the generation of different copies of the same clause, i.e., the computation of several different renamings. This is known as *standardization apart* (Robinson, 1965). In the theory of logic programming, where also mgu substitutions are members of the resolution derivation, this leads to rather complex notions of *standardizing* (Apt, 1990; Ko and Nadel, 1991). Using mgi-c, our formulation can ignore requirement (a) and correctly applies also to clauses containing colliding variables. This is due to operator mgi-c, which deals with formulas containing colliding variables and considers these variables as different. In fact, let C_1, C_2 contain colliding variables. Let ρ_1, ρ_2 be a pair of renamings such that $C_1' = C_1\rho_1$, $C_2' = C_2\rho_2$ and C_1', C_2' satisfy (a), (b), (c), (d) for F, G, θ, and resolvent R. Then Theorem 31 holds again when Z is computed starting from the pair C_1, C_2 as above, while F, G, and θ are selected in the variants C_1', C_2'. In this case, θ is not the mgu of $\{(V_1\ C_2), (C_1\ V_2)\}$ but θ_1 and θ_2 as in (b) of Proposition 30 still exist. Then according to (c) of the same proposition,

$$\text{mgi-c}\{(V_1\ C_2), (C_1\ V_2)\} = (V_1\ C_2)\,\rho_1\mu \sim (V_1\ C_2')\,\theta.$$

Hence, in our formulation, resolution stays entirely within the structure of formulas and applies to the full structure of closed formulas, removing the requirement for standardization apart.

There are several classes of resolvents and correspondingly several resolution procedures that have been proved sound and complete (Chang and Lee, 1974). All of them can be expressed using mgi-c as first class operation in a way similar to that described above.

Finally two words on the use of mgi-c to compute resolvents. As we have seen, mgi-c computes suprema of formulas; thus resolvents are here obtained as suprema of formulas, which are strongly related to the original clauses. This shows that clauses can also be partially ordered so that resolvents are suprema of the resolved clauses. The structure of this ordering depends on the class of resolvents we are looking for. These aspects are currently under development.

6.1. *Resolution in Horn Clause Logic Programming*

In this framework, we are interested not in proving formulas, but rather in knowing the set of terms that, substituted for the variables in the formulas, legitimizes our derivation. This is in fact, the hidden use of substitutions in logic programming. In Horn clause programming languages, the interest is in *answer substitution*, which concerns only the variables of the initial goal clause. Let

- X be the tuple of (distinct) variables occurring in the goal G, and
- $(r\ X)$ be a term with dummy function symbol r and variables in X as arguments, and
- f be an enumeration function from variables to positive integers such that $(r\ X)/f(x) = x$ for each $x \in X$.

Then $(r\ X)\,\theta$ is a presentation of θ restricted to X. We indicate it by $\mathrm{Rep}(\theta, X)$. For the identity substitution ε restricted to X, $\mathrm{Rep}(\varepsilon, X) = (r\ X)$. Moreover, for each $x \in X$, $\mathrm{Rep}(\theta, X)/f(x) = x\theta$. This shows a strong relationship between atomic formulas and restricted substitutions. We now exploit such relationship for the answer substitutions in Horn clause programming.

Given a goal clause, $G = \forall X,\ g_1 \vee \cdots \vee g_n$, let $[G]$ be the sequence of closed atomic formulas $g_1 \cdots g_n\ \mathrm{Rep}(\theta, Y)$, for a substitution θ and vector of variables Y. We require that $R(\theta) \subseteq X \subseteq Y$ hold for the range $R(\theta)$ and vectors X and Y. An initial goal G will be presented by the sequence $g_1 \cdots g_n\ \mathrm{Rep}(\varepsilon, Y)$ for $Y = X = \mathrm{Var}(G)$ and the identity substitution ε. Then we present unit and definite clauses with

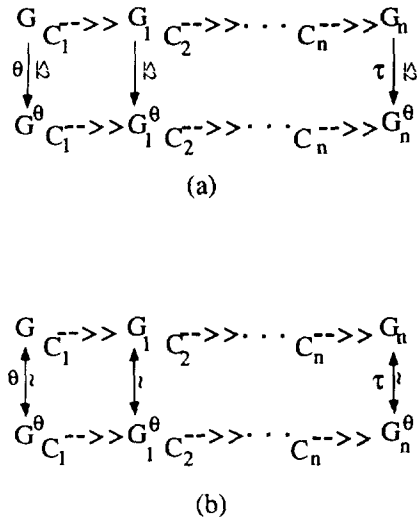

FIG. 4. Lifting and variant lemmas.

sequences as for clauses in general resolution, and present goal clauses as above.

As an example, consider the program $P = \{C1: (A\ Y\ Y) \leftarrow, C2: (B\ 0) \leftarrow\}$ and goal $G = \leftarrow (A\ X\ Z), (B\ Z)$; $[G] = (A\ X\ Z)(B\ Z)(r\ X\ Z)$ and the derivation proceeds in two steps:

$$[G]_{C_1} \rightarrow\!\!\!\rightarrow (B\ X)(r\ X\ X)_{C_2} \rightarrow\!\!\!\rightarrow (r\ 0\ 0).$$

The first step reduces using C_1, while the second one reduces the current goal using C_2. The answer substitution can be extracted from $(r\ 0\ 0)$ and binds the variables X and Z in $(r\ X\ Z)$ to 0.

In this formulation, Horn clause resolution does not require substitutions. Variable bindings are maintained in the goal formula using an additional atom. This makes the theory free from unpleasant and complex requirements like the standardizing techniques [Ko and Nadel, 1991) and also simplifies the theory in some postulates (Apt, 1990). For instance, the *lifting lemma* reduces to well known properties of lattices. Namely, let $G^\theta = G\theta$ for a clause G and a substitution θ. Then $G \leqslant 2\ G^\theta$ and for any C we have $\text{lub}\{G, C\} \leqslant 2\ \text{lub}\{G^\theta, C\}$. Hence τ exists such that $\text{lub}\{G, C\}\ \tau = \text{lub}\{G^\theta, C\}$. This leads to the diagram (a) in Fig. 4. Finally, when θ is a permutation, $G \sim G^\theta$ and the diagram (a) reduces to diagram (b), which is the *variant lemma.* In this case, however, the two sequences are the same in $[\Sigma(P, \mathcal{V})/\sim, \leqslant 2]$ and the variant lemma is no longer relevant.

7. CONCLUSIONS

The original formulation of resolution requires unification, instantiation, renaming, and substitution composition as first class operations, and Σ-terms and substitutions as first class structures. Substitutions have a poor algebraic structure, and need restriction to idempotent substitutions to characterize univocally the most general unifier of open formulas. On closed formulas the situation is even worse, since a most general unifier cannot be characterized at all. Also, substitutions are not satisfactory, from an implementation viewpoint. Most of the implementations of resolution based languages, including WAM (Warren, 1983; Aït-Kaci, 1991), do not provide structures for substitutions, because they are considered too expensive. This causes a rather complex theory and a gap between theory and implementation, which makes correctness hard to prove (Kursawe, 1987; Russinoff, 1992). Similarly, though linear algorithms for unification are well known, none of such implementations involves a unification operation running linearly.

We have defined two distinct operators, mgi-o and mgi-c, as first class operators for open and closed formulas, respectively. Applications of these operators maintain the calculus entirely within the structure of formulas, avoiding substitutions and the related operations of unification and instantiation. We give a new formulation of resolution. Relevant aspects are:

(a) It is based on a more compact structure. Substitutions are eliminated and also the need for some complex techniques of standardizing. Besides, it simplifies domains in abstract interpretations of logic languages (Abramsky and Hankin, 1987).

(b) It suits better from a theoretical point of view. The notion of lub of closed formulas is not affected by variants. Some postulates of the theory are significantly simplified.

(c) It is acceptable from the efficiency point of view. The rules given to compute mgi-o and mgi-c can run linearly as the best unification algorithms. In addition, such implementation naturally copes with laziness.

(d) It provides a direct correspondence between theory and implementation. Based on these structures, a formal derivation of reduction machines for narrowing (Bellia and Occhiuto, 1993) and resolution (Bellia and Occhiuto, 1992) is obtained.

Finally, these structures can be used also for other inference rules that are based on unification, such as narrowing or completion (Bellia and Occhiuto, 1993). This is especially interesting in the framework of languages integrating the functional and the logic paradigm, such as Leaf and Babel (Barbuti *et al.*, 1985; Moreno-Navarro and Rodriguez-Artalejo, 1992).

ACKNOWLEDGMENT

The authors are thankful to the referees for many helpful remarks and suggestions that improved the paper.

Received June 22, 1991; final manuscript received June 2, 1993

REFERENCES

Abramsky, S., and Hankin, C. (1987), "Abstract Interpretation of Declarative Languages," Ellis Horwood, Chichester.

Aït-Kaci, H. (1991), "Warren Abstract Machine: A Tutorial Reconstruction," MIT Press, Cambridge, MA.

Apt, K. R. (1990), Logic programming, *in* "Handbook of Theoretical Computer Science, Vol. B. Formal Models and Semantics" (J. van Leeuwen, Ed.), pp. 493–574, Elsevier, Amsterdam.

Asperti, A., and Martini, S. (1989), Projections instead of variables: A category theoretic interpretation of logic programs, *in* "Proceedings of the 6th International Conference on Logic Programming," pp. 337–352.

Baader, F. (1990), "Unification, Weak Unification, Upper Bound, Lower Bound, Generalization Problems," SEKI-Report SR-90-2, Universität Kaiserlautern.

Barbuti, R., Bellia, M., Levi, G., and Martelli, M. (1985), LEAF: A language which integrates logic, equations and functions, *in* "Logic Programming: Functions, Relations and Equations" (D. DeGroot and G. Lindstrom, Eds.), pp. 201–238, Prentice-Hall, Englewood Cliffs, NJ.

Bellia, M., Bugliesi, M., and Occhiuto, M. E. (1990), Combinatory forms for equational programming: Instances unification and narrowing, *in* "Proceedings of CAAP '90," pp. 42–56, Lecture Notes in Computer Science, Vol. 431, Springer-Verlag, Berlin.

Bellia, M., and Occhiuto, M. E. (1991), Unification: Σ-terms or substitutions? *in* "Atti del 6° Convegno sulla Programmazione Logica," GULP '91, Pisa, pp. 177–189.

Bellia, M., and Occhiuto, M. E. (1992), "Reduction Machines for Logic Programming Based on *C*-Expressions," Technical Report TR-8/92, Università di Pisa.

Bellia, M., and Occhiuto, M. E. (1993), *C*-expressions: A variable free calculus for equational logic programming, *Theoret. Comput. Sci.* **107**, 1993, 209–252.

Chang, C., and Lee, R. C. (1974), "Symbolic Logic and Mechanical Theorem Proving," Academic Press, New York.

Cline, M. P. (1990), "A Linear Parallel-Time Algorithm for *N*-ary (*N*-Statement) Unification," UNIF'90, Leeds.

Eder, E. (1985), Properties of substitutions and unification, *J. Symbolic Comput.* **1**, 31–46.

Falaschi, M., Levi, G., Palamidessi, C., and Martelli, M. (1989), Declarative modeling of the operational behavior of logic languages, *Theoret. Comput. Sci.* **69**, 289–318.

Friedman, D. P., and Wise, D. S. (1976), Cons should not evaluate its arguments, *in* "Automata, Languages and Programming" (S. Michaelson and R. Milner, Eds.), pp. 257–284, Edinburgh Univ. Press.

Herbrand, J. (1930), Recherches sur la Théorie de la Démonstration, Université de Paris. *In* "Ecrits logiques de J. Herbrand," Presses Univ. France, Paris, 1968.

Huet, G. (1972), "Constrained Resolution: A Complete Method for Higher Order Logic," Ph.D. Dissertation, Case Western Reserve University, Cleveland.

Huet, G. (1976), "Résolution d'équations dans les languages d'ordre 1, 2, ..., ω," Ph.D. dissertation, Univ. de Paris VII, Paris.

Huet, G. (1980), Confluent reductions, abstract properties and applications to term rewriting systems, *J. Assoc. Comput. Mach.* **27**, No. 4, 797–821.

Kirchner, C. (1990), "Unification," Academic Press, New York.

Knight, K. (1989), Unification: A multidisciplinary survey, *ACM Comput. Surv.* **21** (1) 93–124.

Ko, H-P., and Nadel, M. E. (1991), Substitution and refutation revisited, *in* "Proceedings of the 8th International Conference on Logic Programming, pp. 679–692.

Kursawe, P. (1987), How to invent a Prolog machine, *New Generation Comput.* **5**, 97–114.

Lassez, J-L., Maher, M. J., and Marriott, K. (1988), Unification revisited, *in* "Foundations of Deductive Databases and Logic Programming" (J. Minker, Ed.), pp. 587–626, Morgan Kaufmann, San Mateo, CA.

Martelli, A., and Montanari, U. (1982), An efficient unification algorithm, *ACM Trans. Programming Languages Systems* **4** (2), 259–282.

Moreno-Navarro, J. J., and Rodriguez-Artalejo, M. (1992), Logic programming with functions and predicates: The language BABEL, *J. Logic Programming* **12** (3), 191–224.

Palamidessi, C. (1990), Algebraic properties of idempotent substitutions, *in* "Proceedings of ICALP'90," pp. 386–399, Lecture Notes in Computer Science, Vol. 443, Springer-Verlag, Berlin.

Paterson, M. S., and Wegman, M. N. (1978), Linear unification, *J. Comput. System Sci.* **16**, 158–167.

Plaisted, D. A. (1984), The occur-check problem in Prolog, *in* "1984, Symposium on Logic Programming," pp. 272–280, IEEE Comput. Soc. Press, Atlantic City, NJ.

Reynolds, J. C. (1970), Transformational system and algebraic structure of atomic formulas, *Mach. Intell.* **5**, 135–151.

Robinson, J. A. (1965), A machine-oriented logic based on the resolution principle, *J. Assoc. Comput. Mach.* **12** (1), 23–41.

Robinson, J. A. (1979), "Logic: Form and Function," Edinburgh Univ. Press, Edinburgh.

Russinoff, D. M. (1992), A verified Prolog compiler for the WARREN Abstract Machine, *J. Logic Programming* **13** (4), 367–412.

Warren, D. H. D. (1983), "An Abstract Prolog Instruction Set," Tech. Rep. 309, SRI International, Menlo Park, CA.